

AD-A197 486

(When Data Entered)

ATTENTION PAGEREAD INSTRUCTIONS
BEFORE COMPLETING FORM

12. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

Ada Compiler Validation Summary Report:
SofTech, Inc., Ada86 Version 1.34, VAX 11/780
and 11/785 Hosts, Intel iAPX 8086 Target.

5. TYPE OF REPORT & PERIOD COVERED
12 June 1987 to 12 June 1988

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

National Bureau of Standards,
Gaithersburg, MD, U.S.A.

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS

National Bureau of Standards,
Gaithersburg, MD, U.S.A.

10. PROGRAM ELEMENT, PROJECT, TASK
AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Ada Joint Program Office
United States Department of Defense
Washington, DC 20301-3081

12. REPORT DATE

12 June 1988

13. NUMBER OF PAGES

64 p.

14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office)

National Bureau of Standards,
Gaithersburg, MD, U.S.A.

15. SECURITY CLASS (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING
SCHEDULE

N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20. If different from Report).

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

DTIC
SELECTED
JUL 13 1988
S D

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada
Compiler Validation Capability, ACVC, Validation Testing, Ada
Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-
1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Ada86 Version 1.34, SofTech, Inc., National Bureau of Standards, VAX 11/780 and 11/785
(hosts) under VAX/VMS 4.5, Intel iAPX 8086 (target) no operating system.
ACVC 1.8.

DD FORM 1473

1 JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

FSV87YRSOF501

Ada* COMPILER
VALIDATION SUMMARY REPORT:
SofTech, Inc.
Ada86 Version 1.34
VAX 11/780 and 11/785 hosts

Target

Intel iAPX 8086

Completion of On-Site Validation:
June 12, 1987

Prepared By:
Software Standards Validation Group
Institute for Computer Sciences and Technology
National Bureau of Standards
Building 225, Room A266
Gaithersburg, MD 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.

Accession For	
NTIS CRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	



*Ada is a registered trademark of the United States Government
(Ada Joint Program Office).

Ada Compiler Validation Summary Report:

Compiler Name: Ada86 Version 1.34

Host Computer:

Target Computer:

VAX 11/780 and 11/785

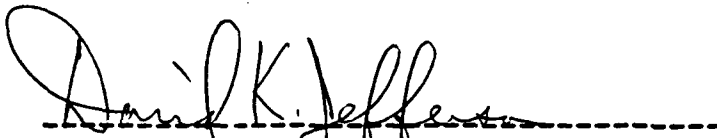
Intel iAPX 8086

under

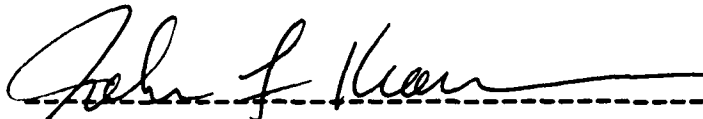
VAX/VMS
4.5

Testing Completed Using ACVC 1.8.

This report has been reviewed and is approved.



Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems Engineering Division
National Bureau of Standards
Gaithersburg, MD 20899-9999



Ada Validation Office
Dr. John F. Kramer
Institute of Defense Analyses
Alexandria VA



Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington, DC

*Ada is a registered trademark of the United States Government
(Ada Joint Program Office).

EXECUTIVE SUMMARY

This Validation Summary Report summarizes the results and conclusions of validation testing performed on the Ada86 Version 1.34 using Version 1.8 of the Ada* Compiler Validation Capability (ACVC).

The validation process includes submitting a suite of standardized tests (the ACVC) as inputs to an Ada compiler and evaluating the results. The purpose is to ensure conformance of the computer to ANSI/MIL-STD-1815A FIPS PUB 119 Ada by testing that it properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by ANSI/MIL-STD-1815A FIPS PUB 119. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, or during execution.

On-site testing was performed June 3, 1987 through June 12, 1987 at Waltham, MA under the auspices of the Software Standards Validation Group, according to Ada Validation Organization policies and procedures. The Ada86 Version 1.34 is hosted on a VAX 11/780 and 11/785 operating under VAX/VMS 4.5.

The results of validation are summarized in the following table:

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	66	861	1026	17	12	46	2028
Failed	0	0	0	0	0	0	0
Inapplicable	3	6	342	0	1	0	352
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

There were 19 withdrawn tests in ACVC Version 1.8 at the time of this validation attempt. A list of these test appears in Appendix D.

Some tests demonstrate that some language features are or are not supported by an implementation. For this implementation, the test determined the following.

- . SHORT_INTEGER is not supported.
- . LONG_INTEGER is supported.
- . SHORT_FLOAT is not supported.
- . LONG_FLOAT is supported.
- . The additional predefined types LONG_INTEGER and LONG_FLOAT are supported.
- . Representation specifications for noncontiguous enumeration representations are supported.
- . The 'SIZE clause is supported.
- . The 'STORAGE_SIZE clause is supported.
- . The 'SMALL clause is supported.
- . Generic unit specifications and bodies can be compiled in separate compilations.
- . Pragma INLINE is supported for procedures.
- . Pragma INLINE is supported for functions.
- . The package SYSTEM is used by package TEXT_IO.
- . Mode IN_FILE is not supported for SEQUENTIAL_IO.
Mode OUT_FILE is not supported for SEQUENTIAL_IO.

- . Instantiation of the package SEQUENTIAL_IO with unconstrained array types is not supported.
- . Instantiation of the package SEQUENTIAL_IO with unconstrained record types with discriminants is not supported.
- . Dynamic creation and resetting of files is not supported for SEQUENTIAL_IO.
- . RESET and DELETE are not supported for SEQUENTIAL and DIRECT_IO.
- . Modes IN_FILE, INOUT_FILE, and OUT_FILE are not supported for SEQUENTIAL_IO.
- . Modes IN_FILE, INOUT_FILE, and OUT_FILE are not supported for DIRECT_IO.
- . Dynamic creation and resetting of files is not supported for DIRECT_IO.
- . Instantiation of package DIRECT_IO with unconstrained array types and unconstrained types with discriminants is not supported.
- . Dynamic creation and deletion of files are not supported.
- . No more than one internal file can be associated with the same external file.
- . An external file associated with more than one internal file cannot be reset.
- . Illegal file names cannot exist.

ACVC Version 1.8 was taken on-site via magnetic tape to Waltham, MA. All tests, except the withdrawn tests and any executable tests that make use of a floating point precision greater than SYSTEM.MAX_DIGITS, were compiled on a VAX 11/780 and 11/785. Class A, C, D, and E tests were executed on the Intel iAPX 8086.

On completion of testing, execution results for Class A, C, D, or E tests were examined. Compilation results for Class B were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link editing results of Class L tests were analyzed for correct detection of errors.

The Software Standards Validation Group identified 2206 of the 2399 tests in Version 1.8 of the ACVC as potentially applicable to the validation of Ada86 Version 1.34. Excluded were 170 tests requiring a floating point precision greater than that supported by the implementation; 4 tests with source lines that were too long; and the 19 withdrawn tests. After the 2206 tests were processed, 178 tests were determined to be inapplicable. The remaining 2028 tests were passed by the compiler.

The Software Standards Validation Group concludes that these results demonstrate acceptable conformance to ANSI/MIL-STD-1815A FIPS PUB 119.

TABLE OF CONTENTS

1.	INTRODUCTION	
1.1	PURPOSE OF THIS VALIDATION SUMMARY REPORT	1-1
1.2	USE OF THIS VALIDATION SUMMARY REPORT	1-2
1.3	RELATED DOCUMENTS	1-3
1.4	DEFINITION OF TERMS	1-3
1.5	ACVC TEST CLASSES	1-4
2.	CONFIGURATION INFORMATION	
2.1	CONFIGURATION TESTED	2-1
2.2	CERTIFICATE	2-2
2.3	IMPLEMENTATION CHARACTERISTICS	2-3
3.	TEST INFORMATION	
3.1	TEST RESULTS	3-1
3.2	SUMMARY OF TEST RESULTS BY CLASS	3-1
3.3	SUMMARY OF TEST RESULTS BY CHAPTER	3-2
3.4	WITHDRAWN TESTS	3-2
3.5	INAPPLICABLE TESTS	3-3
3.6	SPLIT TESTS	3-6
3.7	ADDITIONAL TESTING INFORMATION	3-7
	3.7.1 Prevalidation	3-7
	3.7.2 Test Method	3-7
	3.7.3 Test Site	3-8
A.	COMPLIANCE STATEMENT	A-1
B.	APPENDIX F OF THE Ada STANDARD	B-1
C.	<u>TEST PARAMETERS</u>	C-1
D.	<u>WITHDRAWN TESTS</u>	D-1

CHAPTER 1 INTRODUCTION

This Validation Summary Report describes the extent to which a specific Ada compiler conforms to ANSI/MIL-STD-1815A FIPS PUB 119. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard (ANSI/MIL-STD-1815A FIPS PUB 119). Any implementation-dependent features must conform to the requirements of the Ada Standard. The entire Ada Standard must be implemented, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to ANSI/MIL-STD-1815A FIPS PUB 119, it must be understood that some difference do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other difference between compilers result from limitations imposed on a compiler by the operating systems and by the hardware. All of the dependencies demonstrated during the process of testing this compiler are given in the report. (1CR)

Validation Summary Reports are written according to a standardized format. The report for several different compilers may, therefore, be easily compared. The information in this report is derived from the test results produced during validation testing. Additional testing information is given in section 3.7 and states problems and details which are unique for a specific compiler. The format of a validation report limits variance between reports, enhances readability of the report, and minimizes the delay between the completion of validation testing and the publication of the report.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

The Validation Summary Report documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- . To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- . To attempt to identify any unsupported language constructs required by the Ada Standard
- . To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted under the supervision of the Software Standards Validation Group according to policies and procedures established by the Ada Validation Organization (AVO). Testing was conducted at Waltham, MA and completed on June 12, 1987.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Validation organization may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformances to ANSI/MIL-STD-1815A FIPS PUB 119 other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139
1211 S. Fern, C-107
Washington, DC 20301-3081

or from the Ada Validation Facility (AVF) listed below.

Questions regarding this report or the validation tests should be directed to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard
Alexandria VA 22311

or to:

Ada Validation Facility
Software Standards Validation Group
Institute for Computer Sciences and Technology
National Bureau of Standards
Building 225, Room A266
Gaithersburg, MD 20899

1.3 RELATED DOCUMENTS

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, FEB 1983.
2. Ada Validation Organization: Policies and Procedures, MITRE Corporation, JUN 1982, PB 83-110601.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., DEC 1984.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. A set of programs that evaluates the conformance of a compiler to the Ada language specification, ANSI/MIL-STD-1815A FIPS PUB 119.
Ada Standard	ANSI/MIL-STD-1815A FIPS PUB 119, February 1983.
Applicant	The agency requesting validation.
AVF	Ada Validation Facility. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.
AVO	The Ada Validation Organization. In the content of this report, the AVO is responsible for setting policies and procedures for compiler validations.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	A test for which the compiler generates a result that demonstrates nonconformance to the Ada Standard.
Host	The computer on which the compiler resides.

Inapplicable	A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	A test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that evaluates the conformance of a compiler to a language specification. In the context of this report, the term is used to designate a single ACVC test. The text of a program may be the text of one or more compilations
Withdrawn	A test which has been found to be inaccurate in checking test conformance to the Ada language specification. A withdrawn test has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformance to ANSI/MIL-STD-1815A FIPS PUB 119 is measured using the Ada Compiler Validation Capability (ACVC). The ACVC contains both legal and illegal Ada program structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Legal programs are compiled, linked, and executed while illegal programs are only compiled. Special program units are used to report the results of the legal programs.

Class A tests check that legal Ada programs can be successfully compiled and executed. However, no checks are performed during execution to see if correct object code was generated. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a message indicating that it has passed.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntactical or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NON-APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no requirements placed on a compiler by the Ada Standard for some parameters (e.g., the number of identifiers permitted in a compilation, the number of units in a library, and the number of nested loops in a subprogram body), a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT-APPLICABLE, PASSED or FAILED message when it is compiled and executed. However, the Ada standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report results. It also provides a set of identity functions used to detect some compiler optimization strategies and force computations to be made by the target computer instead of by the compiler on the host computer. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard.

The operation of these units is checked by a set of executable test. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

Some of the conventions followed in the ACVC are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values. Appendix C contains a list of the values used for this validation.

A compiler must correctly process each of the tests in the suite and demonstrate conformance to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and therefore, is not used in testing a compiler. The nonconformant tests are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: Ada86 Version 1.34

Test Suite: Ada Compiler Validation Capability, Version 1.8

Host Computer:

Machine(s): VAX 11/780 and 11/785

Operating Systems: VAX/VMS
4.5

Memory Size: 12 megabytes

Target Computer:

Machine(s): Intel iAPX 8086

Operating System: Bare machine(s)

Memory Size:

Communications Network: DECNET*
Ethernet

*DECNET for this implementation represents the use of VAX 11/780 and VAX 11/785 as hosts

2.2 CERTIFICATE INFORMATION

Base Configuration:

Compiler: Ada86 Version 1.34

Test Suite: Ada Compiler Validation Capability, Version 1.8

Certificate Date: June 12, 1987

Host Computer:

Machine(s): VAX 11/780 and 11/785
Operating System: VAX/VMS
 4.5

Target Computer:

Machine(s): Intel iAPX 8086
Operating System: Bare machine(s)

Communications Network: DECNET
 Ethernet

2.3 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementation to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

- Nongraphic characters.

Nongraphic characters are defined in the ASCII character set but are not permitted in Ada programs, even within character strings. The compiler correctly recognizes these characters as illegal in Ada compilations. The characters are not printed in the output listing. (See test B26005A.)

- Capacities.

The compiler correctly processes compilations containing loop statements nested to 65 levels, block statements nested to 65 levels, procedures nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H, D56001B, D64005E..G and D29002K)

. Universal integer calculations.

An implementation is allowed to reject universal integer calculations having values that exceed `SYSTEM.MAX_INT`. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

Universal real calculations.

When rounding to integer is used in a static universal real expression, the value appears to be rounded towards zero. (See test C4A014A).

Predefined types.

This implementation supports the additional predefined types `LONG_INTEGER` and `LONG_FLOAT` in the package `STANDARD`. (See test C34001D, C34001G.)

Based literals.

An implementation is allowed to reject a based literal with a value exceeding `SYSTEM.MAX_INT` during compilation, or it may raise `NUMERIC_ERROR` during execution. This implementation raises `NUMERIC_ERROR` during execution. (See test E24101A).

Array types.

An implementation is allowed to raise `NUMERIC_ERROR` for an array having a `"LENGTH` that exceeds `STANDARD.INTEGER'LAST` and/or `SYSTEM.MAX_INT`.

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises no exception. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST). (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises no NUMERIC_ERROR. (See test E52103Y.)

In assigning one-dimensional array types, the entire expression appears to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the entire expression does not appear to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Discriminated types.

During compilation, and implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminate constraint. This implementation accepts such subtype indications during compilation. (See test E38104A.)

In assigning record types with discriminants, the entire expression appears to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- . Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index subtype. (See tests C43207A and C43207B).

In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

All choices are not evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

- . Functions.

The declaration of a parameterless function with the same profile as an enumeration literal in the same immediate scope is rejected by the implementation. (See test E66001D.)

- . Representation clauses.

Only 'STORAGE_SIZE for collections is supported. All other length clauses are also accepted but have no affect; the compiler should reject these clauses. The AVO allows this behavior, for the implementor was not advised to change it, and the ACVC does not test the implementation of length clauses in any significant way. (See tests C55B16A, C87B62A..C, and BC1002A.)

- . Generics.

When given a separately compiled generic unit specification, some illegal instantiations, and a body, the compiler rejects the body because of the instantiations. (See tests BC3204C and BC3204D.)

- . Pragma.

The pragma `INLINE` is supported for procedures.
The pragma `INLINE` is supported for functions. (See tests CA3004E and CA3004F.)

- . Input/output.

The package `SEQUENTIAL_IO` cannot be instantiated with unconstrained array types and record types with discriminants. The package `DIRECT_IO` cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (see tests CE2201D, CE2201E, and CE2401D.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

The Software Standards Validation Group identified 2206 of the 2399 tests in Version 1.8 of the Ada Compiler Validation Capability as potentially applicable to the validation of Ada86 Version 1.34. Excluded were 170 tests requiring a floating-point precision greater than that supported by the implementation; 4 tests with source lines that were too long; and the 19 withdrawn tests. After they were processed 178 tests were determined to be inapplicable. The remaining 2028 tests were passed by the compiler.

The Software Standards Validation Group concludes that the testing results demonstrate acceptable conformance to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	<u>A</u>	<u>B</u>	<u>C</u>	<u>D</u>	<u>E</u>	<u>L</u>	
Passed	66	861	1026	17	12	46	2028
Failed	0	0	0	0	0	0	0
Inapplicable	3	6	342	0	1	0	352
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

TEST INFORMATION

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	2	3	4	5	6	7	8	9	10	11	12	14	Total
Passed	98	252	334	244	161	97	137	261	130	32	215	67	2028
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
NA	18	73	86	3	0	0	2	1	0	0	3	166	352
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

3.4 WITHDRAWN TESTS

The following tests have been withdrawn from the ACVC Version 1.8:

B33203C	B37401A	B45116A	B49006A
B4A010C	B74101B	BC3204C	
C32114A	C34018A	C35904A	C41404A
C48008A	C87B50A	C92005A	C940ACA
CA3005A..D (4 tests)			

See Appendix D for the test descriptions.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. For this validation attempt, 352 tests were inapplicable for the reasons indicated:

- . C24113H..K - Line lengths are too long for this implementation
- . C34001D, B52004E, B55B09D, C55B07B -
SHORT_INTEGER is not supported.
- . C34001F, C35702A - SHORT FLOAT is not supported.
- . B86001DT - No additional predefined types are supported.
- . C86001F - Package SYSTEM is used with package TEXT_IO.
- . BC3009A..C - These are ruled inapplicable although these tests are correct and the implementation's behavior on them demonstrates a non-conformity: circularity in generic instantiations is not detected within a generic unit (this violates the Ada Standard 12.3(18) and AI-000328). These tests are allowed to be NA only because the implementer was not informed by the certification body that the similar ruling on the previous validation was subject to review (and thus limited in effect), and the implementer predicated this validation on the previous one.
To prove the implementer's assertion that detection of the circularity these tests check for is indeed made when instantiation outside of a generic unit is made, modified versions of the tests were compiled and the compiler detected all of the intended errors.

. C96005B - DURATION'BASE'FIRST = DURATION'FIRST and
DURATION'BASE'LAST = DURATION'LAST

.These tests are inapplicable because SYSTEM.MAX_ DIGITS
= 15. These tests were:

C24113L through C24113Y (14 tests)
C35705L through C35705Y (14 tests)
C35706L through C35706Y (14 tests)
C35707L through C35707Y (14 tests)
C35708L through C35708Y (14 tests)
C35802L through C35802Y (14 tests)
C45241L through C34524Y (14 tests)
C45321L through C45321Y (14 tests)
C45421L through C45421Y (14 tests)
C45424L through C45425Y (14 tests)
C45521L through C45521Z (15 tests)
C45621L through C45621Z (15 tests)

AE2101C, AE2101H - SEQUENTIAL and DIRECT_IO being
instantiated with unconstrained array
types and record types with
discriminants, without defaults are not
supported.

These tests are inapplicable because this implementation does not support non-file implementations.

AE3101A

EE3102C

CE2102C	CE2102G	CE2104A	CE2104B
CE2104C	CE2104D	CE2105A	CE2106A
CE2107A	CE2107B	CE2107C	CE2107D
CE2107E	CE2107F	CE2108A	CE2108B
CE2108C	CE2108D	CE2109A	CE2110A
CE2110B	CE2110C	CE2111A	CE2111B
CE2111C	CE2111D	CE2111E	CE2111G
CE2111H	CE2201A	CE2201B	CE2201C
CE2201D	CE2201E	CE2201F	CE2204A
CE2204B	CE2210A	CE2401A	CE2401B
CE2401C	CE2401D	CE2401E	CE2401F
CE2404A	CE2405B	CE2406A	CE2407A
CE2408A	CE2409A	CE2410A	CE3102B
CE3103A	CE3104A	CE3107A	CE3108A
CE3108B	CE3109A	CE3110A	CE3111A
CE3111B	CE3111C	CE3111D	CE3111E
CE3112A	CE3112B	CE3114A	CE3114B
CE3115A	CE3203A	CE3208A	CE3301A
CE3301B	CE3301C	CE3302A	CE3305A
CE3402A	CE3402B	CE3402C	CE3402D
CE3403A	CE3403B	CE3403C	CE3403E
CE3403F	CE3404A	CE3404B	CE3404C
CE3405A	CE4305B	CE3405C	CE3405D
CE3406A	CE3406B	CE3406C	CE3406D
CE3407A	CE3407B	CE3407C	CE3408A
CE3408B	CE3408C	CE3409A	CE3409C
CE3409D	CE3409E	CE3409F	CE3410A
CE3410C	CE3410D	CE3410E	CE3410F
CE3411A	CE3412A	CE3413A	CE3413C
CE3602A	CE3602B	CE3602C	CE3602D
CE3603A	CE3604A	CE3605A	CE3605B
CE3605C	CE3605D	CE3605E	CE3606A
CE3606B	CE3704A	CE3704B	CE3704D
CE3704E	CE3704F	CE3704M	CE3704N
CE3704O	CE3706D	CE3706F	CE3804A
CE3804B	CE3804C	CE3804D	CE3804E
CE3804G	CE3804I	CE3804K	CE3804M
CE3805A	CE3805B	CE3806A	CE3806D
CE3806E	CE3905A	CE3905B	CE3905C
CE3905L	CE3906A	CE3906B	CE3906C
CE3906E	CE3906F		

Total of 162 C tests, 1 A test, and 1 E test-- - 164 NA ch.14 tests.

3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of smaller tests that contain the undetected errors. These splits are then compiled and examined. The splitting process continues until all errors are detected by the compiler or until there is exactly one error per split. Any Class A, Class C, or Class E test that cannot be compiled and executed because of its size is split into a set of smaller subtests that can be processed.

Splits were required for 12 Class B tests and 1 Class C tests.

B33301A	B37301J	B44001A
B45205A	B55A01A	B67001B
B67001C	B67001D	B95063A
BC10AEB	BC1202B	BC1202D

C95013A - The text of C95013A had been slightly modified to explicitly spell CALENDAR.TIME wherever type TIME is used. The implementation for the Ada86 toolset has a type TIME defined in package SYSTEM. There is a WITH and USE of both SYSTEM and CALENDAR within this test. An explicit spelling of the type TIME to be used clears any ambiguity. CALENDAR.TIME is what the test is expecting to use. This clarification in the test cases does not alter or impact the objective of the test.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.8 produced by Ada86 Version 1.34 was submitted to the Software Standards Validation Group by the applicant for pre-validation review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests.

3.7.2 Test Method

A test magnetic tape containing ACVC Version 1.8 was taken on-site by the validation team. This magnetic tape contained all tests applicable to this validation as well as all tests inapplicable to this validation except for any Class C tests that require floating-point precision exceeding the maximum value supported by the implementation and tests that exceeded the implementors line length. Tests that were withdrawn from ACVC Version 1.8 were not run. Tests that make use of values that are specific to an implementation were customized before being written to the magnetic tape.

The ACVC B and L tests were compiled and linked on the VAX 11/780 and 11/785 hosts and the results of these tests were analyzed and found to be correct.

The A, C, D, and E tests were executed on the Intel iAPX 8086 target configuration. The results of these tests were examined and found to be in accordance with the Ada Standard.

The magnetic tape was written in the VAX/VMS BACKUP mode and was loaded to disk. Processing began by using command scripts provided by SofTech, Inc..

The development system used to download the executable tests from the VAX host to the Intel targets consisted of:

- Intel Series IV Microprocessor Development System (MDS)
- Intel NDS/VAX II Link (ETHERNET communications software)
- Intel Network Resource Manager (NRM)
- Intel In-Circuit Emulation I2ICE

All executable tests were downloaded and executed in batches of 25 tests to the Intel chip target.

The procedures for executing the tests on each chip target were as follows:

- | | |
|----------------|--|
| V034COICE.VMS | - DCL script to compile, link and export a batch of 25 tests on VAX/VMS (host) |
| Ada86 | - SofTech tools to compile, link and export ACVC tests |
| LNK86 | |
| EXP86 | |
| V034COICE.LOG | - Log file containing results of the compilation, link and export of the 25 tests in the associated .VMS scripts file. |
| C23001A.OBJ | - One of the 25 object files created from exporting the linked container of an individual compiled ACVC test. |
| LOC86 | - Intel tool to generate an absolute object module from the relocatable object module created by the EXP86tool. |
| C23001A.DAT | - Object file to be downloaded and executed on the Intel target. |
| ETHERNET | - Network communication to transfer object modules from the VAX (host) to the Intel Network Resource Manager (NRM). |
| NRM | - Intel Network Resource Manager contains the executable image ready to be loaded and executed on the chip target |
| exec.
image | |
| MDS | - Intel Series IV Microprocessor Development System contains the script file to load and run the executable image invoking internal macro files to use the I2ICE and finally to retain the the output of the transaction into a .raw file. |

I2ICE

iAPX80xxx

xxxxxx.Raw

ETHERNET

xxxxxx.Out

- In-Circuit Emulator contains the macro to execute the test.
- Target chip execution.
- Resort file on NRM
- Upload results file back to VAX for analysis
- Final VAX readable results file.

HOST SYSTEM:

Configurations for SofTech's VAX cluster at the time of TME validation:

VAX CLUSTER NODE NAMES:	SOFT A	SOFT C	SOFT E	SOFT H	SOFT I
OPERATING SYSTEM:	VAX/VMS 4.5	VAX/VMS 4.5	VAX/VMS 4.5	VAX/VMS 4.5	VAX/VMS 4.5
ECC MOS MEMORY:	12M	9.5M	12M	12M	12M
DISK DRIVES:	----- 19/RAB1's CLUSTER-WIDE -----				
ATM/TYPE	(plus) 2/RP06 3/RP06				
SYSTEM DISK DRIVE:	----- 1/RAB1 CLUSTER-WIDE -----				
DUAL PORTED TAPE DRIVE:	1/TA78 and 2/TU78 available CLUSTER-WIDE				
PORTS FOR TERMINALS:	32/(DZ11 and DHU11) per CPU				
AVAILABLE PRINTER: LPM	600	1000	1000	1000	1000

3.7.3 Test Site

The validation team arrived at Waltham, MA on June 3, 1987 and departed after testing was completed on June 12, 1987.

APPENDIX A (of SofTech's Pre-validation material for Ada86 compiler)

SofTech's Compliance Statement for the Ada86 compiler.

APPENDIX A

DECLARATION OF CONFORMANCE

Compiler Implementer: SofTech Inc.
460 Totten Pond Road
Waltham, MA 02254-9197

Ada Validation Facility: National Bureau of Standards (NBS)
Institute for Computer Sciences and Technology (ICST)
Software Standards Validation Group
Building 225, Room A266
Gaithersburg, MD 20899-9999

Ada Compiler Validation Capability (ACVC) Version: 1.8

BASE CONFIGURATION(S)

Base Compiler Name:	Ada86	Version:	1.34
Host Architecture	- ISA: VAX 11/780 - 11/785	OS&VER #:	VAX/VMS 4.5
Target Architecture	- ISA: Intel iAPX 8086	OS&VER #:	(bare machine)

Base Compiler Name:	Ada86	Version:	1.34
Host Architecture	- ISA: VAX 11/780 - 11/785	OS&VER #:	VAX/VMS 4.5
Target Architecture	- ISA: Intel iAPX 80186	OS&VER #:	(bare machine)

Base Compiler Name:	Ada86	Version:	1.34
Host Architecture	- ISA: VAX 11/780 - 11/785	OS&VER #:	VAX/VMS 4.5
Target Architecture	- ISA: Intel iAPX 80286 real mode	OS&VER #:	(bare machine)

Base Compiler Name:	Ada86	Version:	1.34
Host Architecture	- ISA: VAX 11/780 - 11/785	OS&VER #:	VAX/VMS 4.5
Target Architecture	- ISA: Intel iAPX 80286 protected mode	OS&VER #:	(bare machine)

DERIVED COMPILER REGISTRATION EQUIVALENT CONFIGURATION(S)

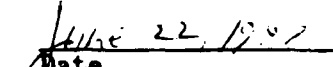
Base Compiler Name:	Ada86	Version:	1.34, 1.59, 1.70
Host Architecture	- ISA: VAX 700 and 8000 Series	OS&VER #:	VAX/VMS 4.5
Target Architecture	- ISA: Intel iAPX 8086	OS&VER #:	(bare machine)
Target Architecture	- ISA: Intel iAPX 80186	OS&VER #:	(bare machine)
Target Architecture	- ISA: Intel iAPX 80286 real mode	OS&VER #:	(bare machine)
Target Architecture	- ISA: Intel iAPX 80286 protected	OS&VER #:	(bare machine)

Base Compiler Name:	Ada86	Version:	1.34, 1.59, 1.70
Host Architecture	- ISA: MicroVAX II	OS&VER #:	MicroVMS 4.5
Target Architecture	- ISA: Intel iAPX 8086	OS&VER #:	(bare machine)
Target Architecture	- ISA: Intel iAPX 80186	OS&VER #:	(bare machine)
Target Architecture	- ISA: Intel iAPX 80286 real mode	OS&VER #:	(bare machine)
Target Architecture	- ISA: Intel iAPX 80286 protected	OS&VER #:	(bare machine)

Implementer's Declaration

I, the undersigned, representing SofTech, Inc., have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that the SofTech Inc. is the owner on record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI-MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.


Implementer's Signature and Title

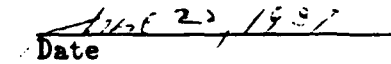

Date

Implementer's Declaration

Owner's Declaration

I, the undersigned, representing SofTech Inc., take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A. I have reviewed the Validation Summary Report for the compilers(s) and concur with the contents.


Owner's Signature and Title


Date

APPENDIX F (of SofTech's Pre-validation material for Ada86 compiler)

APPENDIX F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A).

If an Ada task entry has been equated to a hardware interrupt through an address clause (c.f. LRM 13.5.1), the occurrence of the hardware interrupt in question is interpreted by the RSL as an entry call to the corresponding task entry. The object code generated to implement interrupt entries includes some overhead, since the Ada programmer is allowed to make use of the full Ada language within the accept body for the interrupt entry.

The pragmas described below let the user specify that interrupt entries, and the tasks that contain them, meet certain restrictions. The restrictions speed up the software response to hardware interrupts.

```
Pragma FAST_INTERRUPT_ENTRY (entry_simple_name,  
                             SYSTEM.ENTRY_KIND literal)
```

This pragma specifies that the named task entry has only accept bodies that execute completely with (maskable) interrupts disabled, and that none of these accept bodies performs operations that may potentially lead to task switches away from the accept body.

Pragma INTERRUPT_HANDLER_TASK

This pragma specifies that the task at hand is degenerate in that the whole task body consists of a single loop, which in turn contains one or several accept statements for fast interrupt entries, and which accesses only global variables.

```
Pragma TRIVIAL_ENTRY (entry_simple_name)
```

This pragma specifies that all accept statements for the named entry are degenerate in that their sequence of statements is empty. Moreover, all entry calls to such an entry are conditional entry calls, and they are issued only from within accept bodies for fast interrupt entries.

.....
(2) Implementation-Dependent Attributes
.....

The predefined attribute, X'DISP, is not supported.


```

--#      record
--#          -- This is a segment selector as defined by the iAPX286 hardware.
--#          -- See page 7-11 of the Intel iAPX286 Programmer's Reference Manual.

--#      DESCRIPTOR_INDEX: DESCRIPTOR_TABLE_INDEX;
--#          -- This is an index into either the global or the local
--#          -- descriptor table. The index will select one of the 8 byte
--#          -- descriptors in the table.
--#          -- The table to use is given by the TABLE_INDICATOR field.
--#          -- NOTE:
--#          -- Even if an index is in the proper range, it might not refer
--#          -- to an existing or valid descriptor. See page 7-5 of the
--#          -- Intel iAPX286 Programmer's Reference Manual.

--#      TABLE_INDICATOR: DESCRIPTOR_TABLE_INDICATOR;
--#          -- Whether the index is an index into the global or the local
--#          -- descriptor table;

--#      REQUESTED_PRIVILEGE_LEVEL: PRIVILEGE_LEVEL;
--#          -- The requested privilege level reflects the privilege level of
--#          -- original supplier of the selector. Needed when addresses are
--#          -- passed through intermediate levels. See page 7-14 of the
--#          -- Intel iAPX286 Programmer's Reference Manual.
--#      end record;

--#      for SEGMENT_REGISTER'SIZE use 16;

--#      for SEGMENT_REGISTER use
--#          record
--#              REQUESTED_PRIVILEGE_LEVEL at 0 range 0..1;
--#              TABLE_INDICATOR          at 0 range 2..2;
--#              DESCRIPTOR_INDEX          at 0 range 3..15;
--#          end record;

--#      NULL_SEGMENT: constant SYSTEM.SEGMENT_REGISTER :=
--#          (0, USE_GLOBAL_DESCRIPTOR_TABLE, 0);

--#STOP iAPX286SP

subtype OFFSET_REGISTER is SYSTEM.REGISTER;

type ADDRESS is
    record
        SEGMENT: SYSTEM.SEGMENT_REGISTER;
        OFFSET : SYSTEM.OFFSET_REGISTER;
    end record;

for ADDRESS'SIZE use 32;

for ADDRESS use
    record
        OFFSET at 0 range 0..15;
        SEGMENT at 2 range 0..15;
    end record;
--see[ UM83 4-10, ASM86 6-57,
--      Ada Issue 7]

--#START iAPX86, iAPX186, iAPX286R
    NULL_ADDRESS : constant SYSTEM.ADDRESS := ( 0, 0 );
--#STOP iAPX86, iAPX186, iAPX286R

--#START iAPX286SP

```

```
--# NULL_ADDRESS : constant SYSTEM.ADDRESS := (SYSTEM.NULL_SEGMENT, 0);
--#STOP iAPX286SP
```

```
subtype IO_ADDRESS      is SYSTEM.REGISTER;
```

```
--#START iAPX86, iAPX186, iAPX286R
  type      ABSOLUTE_ADDRESS is range 0..16#FFFFFF#;
  for      ABSOLUTE_ADDRESS'SIZE use 20;
  -- Ada SIZE attribute gives 20, but machine size is 32.
--#STOP iAPX86, iAPX186, iAPX286R
```

```
--#START iAPX286SP
--#  type      ABSOLUTE_ADDRESS is range 0..16#FFFFFFF#;
--#  for      ABSOLUTE_ADDRESS'SIZE use 24;
--#  -- Ada SIZE attribute gives 24, but machine size is 32.
--#STOP iAPX286SP
```

```
type NAME is( VAX780_VMS, iAPX86, iAPX186, iAPX286R, iAPX286SP );
```

```
--#START iAPX86
  SYSTEM_NAME : constant SYSTEM.NAME := ( SYSTEM.iAPX86 );
  --Intel 8086 in real address mode.
--#STOP iAPX86
```

```
--#START iAPX186
--#  SYSTEM_NAME : constant SYSTEM.NAME := ( SYSTEM.iAPX186 );
--#  --Intel 80186 in real address mode.
--#STOP iAPX186
```

```
--#START iAPX286R
--#  SYSTEM_NAME : constant SYSTEM.NAME := ( SYSTEM.iAPX286R );
--#  --Intel 80286 in real address mode.
--#STOP iAPX286R
```

```
--#START iAPX286SP
--#  SYSTEM_NAME : constant SYSTEM.NAME := ( SYSTEM.iAPX286SP );
--#  --Intel 80286 in protected virtual address mode (simple implementation).
--#STOP iAPX286SP
```

```
STORAGE_UNIT: constant := 8;
```

```
--#START iAPX86, iAPX186, iAPX286R
  MEMORY_SIZE : constant := (2**20)-1 ;      -- 1_048_575
--#STOP iAPX86, iAPX186, iAPX286R
```

```
--#START iAPX286SP
--#  MEMORY_SIZE : constant := (2**24)-1 ;      -- 16_777_215
--#STOP iAPX286SP
```

```
MIN_INT : constant := -(2**31) ;      -- -2_147_483_647
MAX_INT : constant := (2**31)-1 ;      -- 2_147_483_647
```

```
MAX_DIGITS : constant := 15;
--Changed from 9 to 15 to match
--change to LONG_FLOAT in package
--STANDARD
```

```
--Note that the Intel 8087 Numeric Data Processor HAS dictated the
--value of MAX_DIGITS.
```

In their App.F. Softtech includes an Ada comment in package SYSTEM for MIN_INT that is incorrect: the value of MIN_INT is 1 less than the comment states (= -(MAX_INT+1), NOT = -(MAX_INT)).

SEE
7E
[GENSE]

3 →

1/2 - 1/2 1/2


```
MAX_MANTISSA: constant := 31;
FINE_DELTA : constant := 4.656_612_873_077_392_578_125E-10; -- 2.0**(-31);
```

```
type INTERRUPT_TYPE_NUMBER is range 0..255;
```

```
--Interrupts having the following Interrupt Type Numbers are specific to the
--iAPX86, iAPX186, and iAPX286 CPUs:
--(Note that the following are declared as CONSTANT universal integers rather
--than CONSTANT SYSTEM.INTERRUPT_TYPE_NUMBERS. This is so that they can be
--used in MACHINE_CODE statements, which require all expressions to be static.
--At least in our implementation, conversions such as
--"MACHINE_CODE.BYTE_VAL( SYSTEM.DISPATCH_CODE_INTERRUPT )" are not considered
--to be static.
```

```
DIVIDE_ERROR_INTERRUPT           : constant := 0;
--Ada semantics dictate that this interrupt must be interpreted as the
--exception NUMERIC_ERROR.
```

```
SINGLE_STEP_INTERRUPT           : constant := 1;
--The non-maskable internal interrupt generated by the CPU after the
--execution of an instruction when the Trap Flag (TF) is set.
```

```
NON_MASKABLE_INTERRUPT         : constant := 2;
--The hardware-generated external interrupt delivered to the CPU via the
--NMI pin. This interrupt can never be disabled by software and can
--penetrate critical regions.
```

```
OVERFLOW_INTERRUPT             : constant := 4;
--Ada semantics dictate that this interrupt must be interpreted as the
--exception NUMERIC_ERROR.
```

```
--Interrupts having the following Interrupt Type Numbers are specific to the
--actual configuration of the iSBC 86/30 board rather than just its CPU:
```

```
--#START iAPX86, iAPX286R, iAPX286SP
RSL_CLOCK_INTERRUPT           : constant := 64;
--This interrupt is reserved for the use of the RSL in maintaining the
--real-time clock and for the support of DELAY statements.
--
--Upper 5 bits, supplied by PIC, are 2#01000#,
--lower 3 bits, derived from PIC input number (IR0), are 2#000#.
--
--By default, this interrupt is the highest in priority.
--
--Assumption: The OUT0 output of the PIT (alias "TIMER 0 INTR") is
--connected to the PIC input IR0.
--#STOP iAPX86, iAPX286R, iAPX286SP
```

```
--#START iAPX186
--# RSL_CLOCK_INTERRUPT           : constant := 18;
--# --This interrupt is reserved for the use of the RSL in maintaining the
--# --real-time clock.
```

```
--# DELAY_EXPIRY_INTERRUPT       : constant := 8;
--# --This interrupt is reserved for the use of the RSL in implementing delays
--# --of less than a full RSL clock cycle.
```

```

--#STOP iAPX186

--#START iAPX86
    NUMERIC_PROCESSOR_INTERRUPT          : constant := 71;
--This interrupt must be interpreted as the exception NUMERIC_ERROR.
--
--Upper 5 bits, supplied by PIC, are 2#01000#,
--lower 3 bits, derived from PIC input number (IR7), are 2#111#.
--
--By default, this interrupt is the lowest in priority.
--
--Assumption: The 8087 interrupt line (alias Math Interrupt or "MINT"), is
--connected to the PIC input IR7.
--#STOP iAPX86

--#START iAPX186
--#    NUMERIC_PROCESSOR_INTERRUPT          : constant := 15;
--# --This interrupt must be interpreted as the exception NUMERIC_ERROR.
--# --
--# --Upper 5 bits, supplied by PIC, are 2#00001#,
--# --lower 3 bits, derived from PIC input number (IR7), are 2#111#.
--# --
--# --By default, this interrupt is the lowest in priority.
--# --
--# --Assumption: The 8087 interrupt line (alias Math Interrupt or "MINT"), is
--# --connected to the PIC input IR7.
--#STOP iAPX186

--#START iAPX286R, iAPX286SP
--#    NUMERIC_PROCESSOR_INTERRUPT          : constant := 16;
--#    --alias Processor Extension Error [PRM Numeric Supplement 1-37]
--#STOP iAPX286R, iAPX286SP

--*** The following RSL internal interrupt type numbers must be changed
--    when the compiler interface has been changed.

--The software interrupt having the following Interrupt Type Number is used
--internally and exclusively by the RSL to check if the current stack
--has enough space:

CHECK_STACK_INTERRUPT          : constant := 48;

--The software interrupt having the following Interrupt Type Number is used
--internally and exclusively by the RSL to effect switching between tasks:

DISPATCH_CODE_INTERRUPT      : constant := 32;

--Interrupts having the following Interrupt Type Numbers (all
--software-generated) are used internally and exclusively by the generated
--code for effecting subprogram entry sequences where there is no SFDD:

ENTER_SUBPROGRAM_WITHOUT_LPP_INTERRUPT : constant := 49;
--The generated code uses this interrupt to effect a subprogram entry
--sequence without a Lexical Parent Pointer.

ENTER_SUBPROGRAM_INTERRUPT     : constant := 50;
--The generated code uses this interrupt to effect a subprogram entry
--sequence with a Lexical Parent Pointer.

```

--Interrupts having the following Interrupt Type Numbers (all software-generated) are used internally and exclusively by the generated code to cause certain Ada exceptions to be forced:

PROGRAM_ERROR_INTERRUPT : constant := 53;
--This interrupt must be interpreted as the exception PROGRAM_ERROR.

CONSTRAINT_ERROR_INTERRUPT : constant := 54;
--This interrupt must be interpreted as the exception CONSTRAINT_ERROR.

NUMERIC_ERROR_INTERRUPT : constant := 55;
--This interrupt must be interpreted as the exception NUMERIC_ERROR.

--Interrupts having the following Interrupt Type Numbers (all software-generated) are used internally and exclusively by the generated code to cause certain RSL services to be invoked:

ALLOCATE_OBJECT_INTERRUPT : constant := 56;
--This interrupt causes an object to be allocated in the heap of the anonymous task.

--Interrupts having the following Interrupt Type Numbers are specific to the Intel iAPX 186 and iAPX 286 CPUs:

BOUND_EXCEPTION_INTERRUPT : constant := 5;
--This interrupt will be interpreted as the exception CONSTRAINT_ERROR.

UNDEFINED_OPCODE_EXCEPTION_INTERRUPT : constant := 6;
--This interrupt will be interpreted as the exception PROGRAM_ERROR.

PROCESSOR_EXTENSION_NOT_AVAILABLE_INTERRUPT: constant := 7;
--This interrupt will be interpreted as the exception PROGRAM_ERROR.

--Intel "reserves" interrupts with Interrupt Type Numbers in the range 0..31, with 32..255 available to the user. We allow the user to equate interrupts in the range 72..103 to entries of task via Ada address clauses. We also allow such use of interrupts 1, 2, and 3, as well as interrupts arriving at

--#START iAPX86, iAPX286R, iAPX286SP
--PIC inputs IR1, IR2, IR3, IR4, IR5, and IR6 (Interrupt Type Numbers 65..70).
--#STOP iAPX86, iAPX286R, iAPX286SP

--#START iAPX186
--# --iAPX186 inputs INT0, INT1, and INT2 (Interrupt Type Numbers 12..14).
--#STOP iAPX186

--The software interrupts having the following Interrupt Type Numbers are used internally and exclusively by the RSL to effect entry to and exit from Innocuous Critical Regions:

ENTER_INNOCUOUS_CRITICAL_REGION_INTERRUPT: constant := 33;

LEAVE_INNOCUOUS_CRITICAL_REGION_INTERRUPT: constant := 34;

--The software interrupts having the following Interrupt Type Numbers are
--defined (and used) by the RSL and can be used by the user:

-- Used to halt the execution of the program from any point.

HALT_INTERRUPT : constant := 36;

END_OF_PROGRAM_INTERRUPT : constant := 37;

-- --\$START IAPX286SP

-- LOAD_TASK_REGISTER_INTERRUPT : constant := 37;

-- CLEAR_TS_FLAG_INTERRUPT : constant := 38;

-- HALT_INTERRUPT : constant := 39;

-- --\$STOP IAPX286SP

pragma PAGE;

--The enumeration literals of type ENTRY_KIND distinguish between entries of
--software tasks and interrupt entries, and identify different varieties of
--the latter when used as the second argument in a FAST_INTERRUPT_ENTRY
--pragma:

type ENTRY_KIND is

(

--ORDINARY_INTERRUPT_ENTRY--

ORDINARY_INTERRUPT_ENTRY,

--This is not a Fast Interrupt Entry. It is invoked by an interrupt
--other than NMI. This entry may be called by a software task as
--well as by interrupt.

--If an interrupt is equated to an entry by means of an address
--clause, and the FAST_INTERRUPT_ENTRY pragma is not given for that
--entry, the entry will be treated as an ORDINARY_INTERRUPT_ENTRY by
--default.

--When this kind of interrupt entry occurs, the state of the 8087
--Numeric Data Processor will always be saved as part of the context
--of the interrupted task, because the normal task-switching
--mechanism will attempt to restore it before resuming the
--interrupted task.

-- FAST INTERRUPT ENTRIES --

--Prompt Interrupt Entry:

PROMPT

--This is a Fast Interrupt Entry, invoked by an interrupt other than
--NMI or Single Step, whose accept body receives control after an
--interrupt more quickly than an ordinary interrupt entry but more
--slowly than a Quick or a Non-Maskable Interrupt Entry. The accept
--body may make conditional entry calls to entries that have been

--declared to be Trivial Entries by means of the pragma
--TRIVIAL_ENTRY.

--When this kind of interrupt entry occurs, the state of the 8087
--Numeric Data Processor will always be saved as part of the context
--of the interrupted task, because the normal task-switching
--mechanism will attempt to restore it before resuming the
--interrupted task.

--Note: In the following constant names, "NDP" stands for "Numeric Data
--Processor," i.e., the Intel 8087.

--Quick Interrupt Entries:

SIMPLE_QUICK

--This is a Quick Interrupt Entry, invoked by an interrupt other than
--NMI or Single Step, whose accept body makes no entry calls.

NO_NDP_SIMPLE_QUICK

--This is a Quick Interrupt Entry, invoked by an interrupt other than
--NMI or Single Step, whose accept body makes no entry calls.

--It differs from SIMPLE_QUICK only in that the state of the 8087
--Numeric Data Processor is neither saved nor restored during
--interrupt delivery.

SIGNALLING_QUICK

--This is a Quick Interrupt Entry, invoked by an interrupt other than
--NMI or Single Step, whose accept body may make conditional entry
--calls to entries that have been declared to be Trivial Entries by
--means of the pragma TRIVIAL_ENTRY.

--When this kind of interrupt entry occurs, the state of the 8087
--Numeric Data Processor will always be saved as part of the context
--of the interrupted task, because the normal task-switching
--mechanism will attempt to restore it before resuming the
--interrupted task.

--Non-Maskable Interrupt Entries:

NON_MASKABLE

--This is a Non-Maskable Interrupt Entry invoked only by NMI whose
--accept body makes no entry calls.

NO_NDP_NON_MASKABLE

--This is a Non-Maskable Interrupt Entry invoked only by NMI whose
--accept body makes no entry calls.

--It differs from NON_MASKABLE only in that the state of the 8087
--Numeric Data Processor is neither saved nor restored during
--interrupt delivery.

);

pragma PAGE;

```

-----
-- NOTE: Be sure to compute TICK and TICKS_PER_DAY by hand, as the roundoff --
-- errors introduced in computer arithmetic are unacceptably inaccurate.  --
-----

```

```

--$START iAPX86
--If one loaded the Programmable Interval Timer (PIT) clock counter with the
--shortest possible delay, namely 1, TICK is the amount of time, in seconds,
--which would pass between the loading and the interrupt which the PIT would
--issue upon counting down and reaching zero.

```

```

    TICK      : constant := 6.510_416_666_666_666_666_667E-6;
--roughly 6.5 microseconds
--$STOP iAPX86

```

```

--$START iAPX186
--$ --For the system clock counter of the iAPX186's Internal Timer Unit, TICK is
--$ --the amount of time, in seconds, that it takes to count from 0 to 1.

--$ --IMPORTANT: The iSBC 186/03A runs at 8 MHz, and its Internal Timer Unit's
--$ --base clock rate is 8 MHz divided by four, or 2 MHz.
--$ --Therefore one counter tick = 1 sec. / 2_000_000 = 0.000_000_5 sec.
--$ --One major clock cycle = 2**16 * one counter tick
--$ --                        = 65_536 * 0.000_000_5 sec.
--$ --                        = 0.032_768 sec.
--$ --We would like a greater time interval between counter interrupts used for
--$ --timekeeping. In fact, we would like about one second, or as close as
--$ --possible. This means that we must prescale our system clock counter.
--$ --
--$ --To find prescale factor, solve for X:
--$ -- X * one major clock cycle = 1 second
--$ -- X * 0.032_768 sec.         = 1 sec.
--$ -- X                          = 1 / 0.032_768
--$ -- X                          = 30.517_578_125
--$ -- X                          ~ = 30
--$ --
--$ --So SYSTEM.TICK = a prescaled counter tick
--$ --                = 30 * 0.000_000_5 sec.
--$ --                = 0.000_015 sec.
--$ --and a prescaled major clock cycle = 2**16 * one prescaled counter tick
--$ --                                = 65_536 * 0.000_015 sec.
--$ --                                ~ = 0.983 sec.
--$ --
--$ --There are 66_666 + 2/3 ticks in a second.
--$ --The number of ticks per second must be used to calculate the values of the
--$ --ADA_RSL constants CLOCK_TICKS_PER_DAY, TICKS_PER_HALF_DAY, and INT_CHUNK_
--$ --RAW_TIME.

```

```

--$ TICK      : constant := 0.000_015; --15 microseconds
--$STOP iAPX186

```

```

--$START iAPX286SP, iAPX286R
--$ --If one loaded the Programmable Interval Timer (PIT) clock counter with the
--$ --shortest possible delay, namely 1, TICK is the amount of time, in seconds,
--$ --which would pass between the loading and the interrupt which the PIT would
--$ --issue upon counting down and reaching zero.

--$ --The CLK0 input to the 8254 PIT on the iSBC 286/10 is 1.23 MHz.
--$ --So one counter 0 tick = 1 sec. / 1_230_000 = 0.0000_00813_00813... sec.
--$ --One major clock cycle = 2**16 * one counter tick

```

```

--# --                                     = 65_536 * 0.0000_00813_00813_... sec.
--# --                                     = 0.0535 sec.
--# --
--# --There are 1_230_000 (in hex, 16#0012_C4B0#) ticks in a second if
--# --is not prescaled.

--# --The maximum recommended value of the smallest delay duration (LRM 9.6) is
--# --50 microseconds. This will give the lowest possible frequency of timer
--# --interrupts. To achieve this, another counter is needed as a prescaler. The
--# --prescale factor (X) is calculated as follows.
--# --    X = 0.0000_5 / One counter 0 tick
--# --    X = 0.0000_5 / 0.0000_00813_00813_....
--# --    X = 61.5
--# --    X = 61 (nearest rounded off value)
--# --Therefore SYSTEM.TICK = 61 * counter 0 tick
--# --                                     = 61 * 0.0000_00813_00813_... sec.
--# --                                     = 0.0000_49593_49593_49593_... sec.
--# --                                     = 49.593_49593_49593_49593_... microseconds
--# --One major clock cycle = 2**16 * SYSTEM.TICK
--# --                                     = 65_536 * 0.0000_49593_49593_49593_... second
--# --                                     = 3.2501_59349_59349_59349_... seconds

--# TICK      : constant := 0.0000_49593_49593_49593; --about 49.59 microseconds
--# TICKS_PER_SECOND : constant := 20163.93442_62209_52836_06557;--approximate
--# --TICKS_PER_SECOND must be used to calculate (by hand!) the values of the
--# --ADA_RSL constants CLOCK_TICKS_PER_DAY, TICKS_PER_HALF_DAY, and INT_CHUNK_
--# --RAW_TIME.

--#STOP iAPX286SP, iAPX286R

type TIME is private;
NULL_TIME : constant TIME;

type DIRECTION_TYPE is( AUTO_INCREMENT, AUTO_DECREMENT );
type PARITY_TYPE      is( ODD, EVEN );

type FLAGS_REGISTER is
  record

    --#START iAPX286SP
--# NESTED_TASK      : BOOLEAN      := FALSE;
--# IO_PRIVILEGE_LEVEL : NATURAL range 0..3 := 1;
--#STOP iAPX286SP

    OVERFLOW : BOOLEAN      := FALSE;
    DIRECTION : SYSTEM.DIRECTION_TYPE := SYSTEM.AUTO_INCREMENT;
    INTERRUPT : BOOLEAN      := TRUE;
    TRAP      : BOOLEAN      := FALSE;
    SIGN      : BOOLEAN      := FALSE;
    ZERO      : BOOLEAN      := TRUE; --nihilistic view
    AUXILIARY : BOOLEAN      := FALSE;
    PARITY    : SYSTEM.PARITY_TYPE := SYSTEM.EVEN;
    CARRY     : BOOLEAN      := FALSE;

  end record;

for FLAGS_REGISTER use
  record

```

```

--#START iAPX286SP
--# NESTED_TASK      at 0 range 14..14;
--# IO_PRIVILEGE_LEVEL at 0 range 12..13;
--#STOP iAPX286SP

```

```

OVERFLOW      at 0 range 11..11;
DIRECTION     at 0 range 10..10;
INTERRUPT     at 0 range 9.. 9;
TRAP          at 0 range 8.. 8;
SIGN          at 0 range 7.. 7;
ZERO          at 0 range 6.. 6;
AUXILIARY     at 0 range 4.. 4;
PARITY        at 0 range 2.. 2;
CARRY         at 0 range 0.. 0;
end record;

```

NORMALIZED_FLAGS_REGISTER : constant SYSTEM.FLAGS_REGISTER :=

```

(
  --#START iAPX286SP
  --# NESTED_TASK      => FALSE,
  --# IO_PRIVILEGE_LEVEL => 1,
  --#STOP iAPX286SP

  OVERFLOW => FALSE,
  DIRECTION => SYSTEM.AUTO_INCREMENT,
  INTERRUPT => TRUE,
  TRAP      => FALSE,
  SIGN      => FALSE,
  ZERO      => TRUE,  --nihilistic view
  AUXILIARY => FALSE,
  PARITY    => SYSTEM.EVEN,
  CARRY     => FALSE
);

```

subtype PRIORITY is INTEGER range 1..15;

UNRESOLVED_REFERENCE: exception;
 SYSTEM_ERROR : exception;

--see Appendix 30 of A-spec

```

function EFFECTIVE_ADDRESS
  ( A: in SYSTEM.ADDRESS
  )
return SYSTEM.ABSOLUTE_ADDRESS;

```

```

--PURPOSE:
-- This function, written in ASM86, returns the 20-bit effective address
-- specified by the segment/offset register pair A.
pragma INTERFACE( ASM86, EFFECTIVE_ADDRESS );

```

```

function FAST_EFFECTIVE_ADDRESS
-- ( A: in SYSTEM.ADDRESS
--   --found in DX (segment part) and AX (offset part), NOT on stack
-- )
return SYSTEM.ABSOLUTE_ADDRESS;
--in DX:AX;

```



```

--PURPOSE:
-- This function, written in ASM86, returns the 20-bit effective address
-- specified by the segment/offset register pair DX:AX.
-- This function is intended for use by ASM routines. It does not observe
-- Ada calling conventions and therefore does not make a null SFDD. It
-- does save and later restore all those registers that it uses
-- internally.
pragma INTERFACE( ASM86, FAST_EFFECTIVE_ADDRESS );

```

```

function TWOS_COMPLEMENT_OF
( W: in SYSTEM.WORD
)
return SYSTEM.WORD;

```

```

--PURPOSE:
-- This function, written in ASM86, returns the two's complement of the
-- given argument.
--ASSUMPTIONS:
-- 1) CRITICAL REGION INFORMATION:
-- This procedure makes no assumptions about critical regions.
-- It neither enters nor leaves a critical region.
pragma INTERFACE( ASM86, TWOS_COMPLEMENT_OF );

```

```

procedure ADD_TO_ADDRESS
( ADDR : in out SYSTEM.ADDRESS;
  OFFSET: in SYSTEM.OFFSET_REGISTER );

```

```

--PURPOSE:
-- This procedure, written in ASM86, adds OFFSET to the offset part of
-- ADDR. If overflow occurs, NUMERIC_ERROR is raised.
--SIDE EFFECTS:
-- Raising of NUMERIC_ERROR.
pragma INTERFACE( ASM86, ADD_TO_ADDRESS );

```

```

procedure SUBTRACT_FROM_ADDRESS
( ADDR : in out SYSTEM.ADDRESS;
  OFFSET: in SYSTEM.OFFSET_REGISTER );

```

```

--PURPOSE:
-- This procedure, written in ASM86, subtracts OFFSET from the offset part
-- of ADDR. If underflow occurs, NUMERIC_ERROR is raised.
--SIDE EFFECTS:
-- Raising of NUMERIC_ERROR.
pragma INTERFACE( ASM86, SUBTRACT_FROM_ADDRESS );

```

```

function INTERRUPT_TYPE_NUMBER_OF
( A : in SYSTEM.ADDRESS
)
return SYSTEM.INTERRUPT_TYPE_NUMBER;

```

```

--PURPOSE:
-- This function, written in ASM86, returns the Interrupt Type Number that
-- uniquely identifies the interrupt whose interrupt vector is located at
-- the specified address. If this address is not the address of an
-- interrupt vector, CONSTRAINT_ERROR is raised.
--SIDE EFFECTS:

```

```

-- Raising of CONSTRAINT_ERROR.
pragma INTERFACE( ASM86, INTERRUPT_TYPE_NUMBER_OF );

procedure GET_ADDRESS_FROM_INTERRUPT_TYPE_NUMBER
( A : out SYSTEM.ADDRESS;
  ITN : in SYSTEM.INTERRUPT_TYPE_NUMBER
);

--PURPOSE:
-- This procedure, written in ASM86, returns the address of the interrupt
-- vector numbered ITN.
pragma INTERFACE( ASM86, GET_ADDRESS_FROM_INTERRUPT_TYPE_NUMBER );

function GREATER_THAN
( A1 : in SYSTEM.ADDRESS;
  A2 : in SYSTEM.ADDRESS
)
return BOOLEAN;

--PURPOSE:
-- This function, written in ASM86, returns the value of the expression
-- A1 > A2;
pragma INTERFACE( ASM86, GREATER_THAN );

function MINUS
( A1 : in SYSTEM.ADDRESS;
  A2 : in SYSTEM.ADDRESS
)
return LONG_INTEGER;

--PURPOSE:
-- This function, written in ASM86, returns the signed value of A1 - A2.
pragma INTERFACE( ASM86, MINUS );

function ">"
( A1 : in SYSTEM.ADDRESS;
  A2 : in SYSTEM.ADDRESS
)
return BOOLEAN renames SYSTEM.GREATER_THAN;

function "--"
( A1 : in SYSTEM.ADDRESS;
  A2 : in SYSTEM.ADDRESS
)
return LONG_INTEGER renames SYSTEM.MINUS;

-- procedure ADJUST_FOR_UPWARD_GROWTH
-- ( OLD_ADDRESS : in SYSTEM.ADDRESS;
--   ADJUSTED_ADDRESS: out SYSTEM.ADDRESS );
-- Transforms the given SYSTEM.ADDRESS into a representation yielding
-- the same effective address, but in which the SEGMENT component is
-- as large as possible.

```

```
-- procedure ADJUST_FOR_DOWNWARD_GROWTH
--   ( OLD_ADDRESS      : in SYSTEM.ADDRESS;
--     ADJUSTED_ADDRESS: out SYSTEM.ADDRESS );
--   Transforms the given SYSTEM.ADDRESS into a representation yielding
--   the same effective address, but in which the OFFSET component is as
--   large as possible.

--private

-- pragma INTERFACE( ASM86, ADJUST_FOR_UPWARD_GROWTH );
-- pragma INTERFACE( ASM86, ADJUST_FOR_DOWNWARD_GROWTH );

private

type LONG_CYCLE is array(1..3)of SYSTEM.WORD;
pragma PACK( LONG_CYCLE ); --Make this type occupy 64 bits.

type TIME is --This may be viewed as a single 64-bit integer
record
    --representing a quantity of SYSTEM.TICKS.
    CYCLES : LONG_CYCLE;
    TICKS   : SYSTEM.WORD;
end record;

for TIME use record
    CYCLES at 0 range 0..47;
    TICKS  at 6 range 0..15;
end record;

--A TIME variable may be viewed as a 64-bit integer, or as a record with a
--more significant CYCLES part and a less significant TICKS part. Whenever
--the TICKS part is incremented, the addition may carry over into the
--adjacent CYCLES part.
--
--Storage layout of a variable of type TIME:
--
--                                     increasing addresses
--                                     ----->
--
-- +-----+-----+-----+-----+
-- | CYCLES(1) | CYCLES(2) | CYCLES(3) | TICKS   |
-- +-----+-----+-----+-----+
--
-- \_____/
--    one word

NULL_TIME : constant TIME := ( (OTHERS => 0), 0 );

end SYSTEM;
```


Are permitted for N in 2..32, provided the representations and the SIZE conform to the relationship specified above, or else for N in 1..31, provided that the internal representation of T'FIRST ≥ 0 and the representation of T'LAST $= 2^{**}(T'SIZE) - 1$.

For components of enumeration types within packed composite objects, the smaller of the default stand-alone SIZE and the SIZE from a length specification is used.

In accordance with the rules of Ada, and the implementation of package STANDARD, enumeration representation on types derived from the predefined type BOOLEAN are not accepted, but length specifications are accepted.

Record Representation Clause

A length specification of the form

for T'SIZE use N;

Will cause arrays and records to be packed, if required, to accommodate the length specification.

The PACK pragma may be used to minimize wasted space between components of arrays and records. The pragma causes the type representation to be chosen such that storage space requirements are minimized at the possible expense of data access time and code space.

A record type representation specification may be used to describe the allocation of components in a record. Bits are numbered 0..7 from the right. (Bit 8 starts at the right of the next higher-numbered byte.)

The alignment clause of the form:

at mod N

can specify alignment of 1 (byte) or 2 (word).

TEXT_IO Package

---* PACKAGE SPECIFICATION FOR TEXT_IO

The Specification of the Package TEXT_IO contains the following
(implementation specific) definitions in addition to those specified
in 14.3.10 of the LRM:

-- Copyright (C) 1986, SofTech, Inc.

with SYSTEM, IO_DEFS;

---* PURPOSE:

--1 This package provides input and output services for textual files

--1

--1 including creation, deletion, opening, and closing of said files.

--1 This package is as specified in the Ada Reference Manual (1982).

--1

--1 And here a word about primary and secondary routines. A primary routine

--1 is always visible outside the package. If it references a file, it will

--1 attempt to gain exclusive access to that file descriptor. (The term

--1 "exclusive access" is used with regard to tasks.) All modifications or

--1 tests on file descriptor FIELDS must be made only if the current task

--1 has exclusive access to that descriptor. In every case where a primary

--1 routine gains exclusive access to a file descriptor, that routine must

--1 release the file descriptor BEFORE exiting. Primary routines may call

--1 primary or secondary routines. Secondary routines are never visible

--1 outside the package. If a secondary routine references a file descriptor,

--1 that routine assumes exclusive access for that descriptor. Secondary

--1 routines may only call other secondary routines. All calls to BASIC_IO

--1 for reading or writing are made by secondary routines. All other

--1 BASIC_IO calls are made by primary routines.

TYPE count IS RANGE 0 .. integer'LAST;

SUBTYPE field IS integer RANGE 0 .. integer'LAST;

-----P R I V A T E -----

PRIVATE

buffer_length : CONSTANT

--#START VAX780_VMS

--1 := 256;

--#STOP VAX780_VMS

--#START IAPX86, IAPX186, IAPX286R, IAPX286SP

--1 := 80;

--#STOP IAPX86, IAPX186, IAPX286R, IAPX286SP

TYPE char_buffer IS ARRAY (io_defs.data_length_int
RANGE 1 .. buffer_length) of character;

```

TYPE file_rec IS -- common file state description; actual FILE_TYPE
RECORD          -- declarations will be access types to this record.

```

```

--#START VAX780_VMS
--# external_name : string_util.var_string_rec ;
--# files_class   : io_defs.file_class_enu := io_defs.fc_text;
--# interactive   : boolean := false;
--# end_info      : boolean := false;
--#STOP VAX780_VMS

```

```

strm_ptr      : io_defs.stream_id_prv;
f_mode        : file_mode;
curr_col      : count := 1;
curr_line     : count := 1;
curr_page     : count := 1;
eoln_found    : boolean := false;
eop_found     : boolean := false;
eof_found     : boolean := false;
line_len      : count := unbounded;
page_len      : count := unbounded;
max_rec_length : count := buffer_length;
curr_rec_length : io_defs.data_length_int := 0;
text_index    : io_defs.data_length_int := 0;
text_buf      : char_buffer;
next_rec_length : io_defs.data_length_int := 0;
next_buf      : char_buffer;
END RECORD;

```

```

max_line_length : CONSTANT :=

```

```

--#START VAX780_VMS
--# 255;
--#STOP VAX780_VMS

```

```

--#START iAPX86, iAPX186, iAPX286R, iAPX286SP
80;
--#STOP iAPX86, iAPX186, iAPX286R, iAPX286SP

```

```

--#START VAX780_VMS
--# max_filename_length : CONSTANT := 20;
--#STOP VAX780_VMS

```

```

TYPE file_type IS ACCESS file_rec;

```

```

-----
std_input  : file_type;      -- the standard and current file descriptors
std_output : file_type;      -- should not be visible to the user except
curr_input : file_type;      -- through the provided procedure (see above).
curr_output : file_type;

```

```

-- Define file marker values.

```

```

line_term  : CONSTANT character := ASCII.LF;
page_term  : CONSTANT character := ASCII.PF;      -- form feed (ctrl-L) (16#0C#)
file_term  : CONSTANT character := ASCII.SUB;     --          (ctrl-Z) (16#1A#)
null_strm  : CONSTANT := 0;                       -- null stream pointer value

```

```

END text_io;

```


LOW_LEVEL_IO

Include either the LOW_LEVEL_IO package specification or the
following sentence:

Low-level input-output is not provided.

-- Copyright (C) 1986, SofTech, Inc.

with SYSTEM; use SYSTEM;

--* PACKAGE SPECIFICATION FOR LOW_LEVEL_IO

--* PURPOSE:

--& To support the programming of devices that can be accessed through ports
--& in the memory space and the I/O space of the iAPX186. Specific devices
--& or device types that cannot be assumed to be present in all iAPX186-based
--& targets should be supported by specific packages (e.g., MPSC).
--

pragma PAGE; -- In package LOW_LEVEL_IO

--* SPECIFICATION:

package LOW_LEVEL_IO is

--Support for I/O-mapped input and output:

```
procedure SEND_CONTROL ( DEVICE : in IO_ADDRESS; DATA : in out BYTE );  
procedure SEND_CONTROL ( DEVICE : in IO_ADDRESS; DATA : in out WORD );  
procedure RECEIVE_CONTROL( DEVICE : in IO_ADDRESS; DATA : in out BYTE );  
procedure RECEIVE_CONTROL( DEVICE : in IO_ADDRESS; DATA : in out WORD );
```

--Support for memory-mapped input and output:

```
procedure SEND_CONTROL ( DEVICE : in ADDRESS; DATA : in out BYTE );  
procedure SEND_CONTROL ( DEVICE : in ADDRESS; DATA : in out WORD );  
procedure RECEIVE_CONTROL( DEVICE : in ADDRESS; DATA : in out BYTE );  
procedure RECEIVE_CONTROL( DEVICE : in ADDRESS; DATA : in out WORD );
```

end LOW_LEVEL_IO;


```

--          ~ 3.40_282_347E+38
--      float'machine_radix = 2
--      float'machine_mantissa = 24
--      float'machine_emax = 127
--      float'machine_emin = -126
--      float'machine_rounds = true
--      float'machine_overflows = true

      TYPE long_float IS DIGITS 15 RANGE
- 2#1.111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1#E+1023
.. 2#1.111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1#E+1023;

--      Type long_float is realized using the Intel machine type LONG REAL.
--      LONG REAL provides 53 bits of mantissa (one bit is implied),
--      and it provides 11 bits for a biased exponent. However only the values
--      1..2046 are exponents of normalized numbers. The bias is 1023, so the
--      exponent range is -1022..1023.
--      This leads to the following attributes for the type float:
--
--      long_float'digits = 15 [LRM 3.5.7, 3.5.8]
--      long_float'mantissa = 51 [LRM 3.5.7, 3.5.8]
--      long_float'emax = 204 [LRM 3.5.8]
--      long_float'epsilon = 2.0 ** (-50) [LRM 3.5.8]
--                          = 16#0.400_000_000_000_00#E-12
--                          ~ 8.88_178_197_001_254E-16
--      long_float'small = 2.0 ** (-205) [LRM 3.5.8]
--      = 2#1.000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0#E-205
--                          = 16#0.800_000_000_000_00#E-51
--                          ~ 1.94_469_227_433_161E-16
--      long_float'large = (2.0 ** 204) * (1.0 - 2.0 ** (-51)) [LRM 3.5.8]
--      = 2#1.111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1110_0#E+204
--                          = 16#0.FFF_FFF_FFF_FFF_E0#E+51
--                          ~ 2.57_110_087_081_438E+61
--      long_float'safe_emax = 1023 [LRM 3.5.7, 3.5.8]
--      long_float'safe_small = (2.0 ** (-1022)) [LRM 3.5.7]
--      = 2#1.000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0#E-1022
--                          = 16#4.000_000_000_000_00#E-256
--                          ~ 2.22_507_385_850_720E-308
--      long_float'safe_large = (2.0 ** 1024) * (1.0 - 2.0 ** (-51)) [LRM 3.5.7]
--      = 2#1.111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1110_0#E+1023
--                          = 16#0.FFF_FFF_FFF_FFF_C#E+256
--                          ~ 1.79_768_713_486_232E+308
--      long_float'first = -long_float'last
--      long_float'last
--      = 2#1.111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1#E+1023
--                          = 16#0.FFF_FFF_FFF_FFF_F#E+256
--                          ~ 1.79_768_713_486_232E+308
--
--      long_float'machine_radix = 2
--      long_float'machine_mantissa = 53
--      long_float'machine_emax = 1023
--      long_float'machine_emin = -1022
--      long_float'machine_rounds = true
--      long_float'machine_overflows = true

```

FOR character'SIZE USE 8;

TYPE duration IS DELTA 2.0 ** (-14) RANGE -131_072.0 .. 131_072.0 ;

END standard;

[illegible]

As SEQUENTIAL_IO and DIRECT_IO are not supported on the target(s), there are no file name conventions on the target configuration(s).

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension. TST in its file name. Actual values to be substituted are identified by names that begin with a dollar sign. A value is substituted for each of these names before the test is run. The values used for this validation are given below.

<u>Name and Meeting</u>	<u>Value</u>
\$BIG_ID1 Identifier of size MAX_IN_LEN with varying last character.	119 A's followed by a B
\$BIG_ID2 Identifier of size MAX_IN_LEN with varying last character.	119 A's followed by a C
\$BIG_ID3 Identifier of size MAX_IN_LEN with varying middle character.	59 A's followed by a 3 followed by 60 A's
\$BIG_ID4 Identifier of size MAX_IN_LEN with varying middle character.	59 A's followed by a 4 followed by 60 A's
\$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is MAX_IN_LEN characters long.	117 0's followed by 298

<u>Name and Meaning</u>	<u>Value</u>
\$BIG_REAL_LIT A real literal that can be either of floating or fixed point type, has value 690.0, and has enough leading zeroes to be MAX_IN_LEN characters long.	114 0's followed by 69.0E1
\$BLANKS Blanks of length MAX_IN_LEN - 20	100
\$CNT_LAST Value of CNT'LAST in TEXT_IO package.	32767
\$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	""abcdefghijklmnopqrstuvwxyz !\$?@[\]^'_{}~""
\$FIELD_LAST Value of FIELD'LAST in TEXT_IO package.	32767
\$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters or is too long.	X)]!@\$^&~Y
\$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	XYZ*
\$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION	100_000.0
\$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST.	10_000_000.0
\$ILLEGAL_EXTERNAL_FILE_NAME Illegal external file name.	BAD-CHARACTER*^
\$ILLEGAL_EXTERNAL_FILE_NAME2 Illegal external file names.	MUCH-TOO-LONG-NAME-FOR-A-FILE

<u>Name and Meaning</u>	<u>Value</u>
\$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-32768
\$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	32767
\$LESS_THAN_DURATION A universal real value that lies between DURATION"BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-100_000.0
\$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less then DURATION'BASE'FIRST.	-10_000_000.0
\$MAX_DIGITS Maximum digits supported for floating-point types.	15
\$MAX_IN_LEN Maximum input line length permitted by the implementation.	120
\$NAME A name of predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER,	LONG_LONG_INTEGER
\$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	8#37777777772#
\$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non-ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. When testing was performed, the following 19 tests had been withdrawn at the time of validation testing for the reasons indicated:

- . B4A010C: The object_declaration in line 18 follows a subprogram body of the same declarative part.
- . BC3204C: The file BC3204C4 should contain the body for BC3204C0 as indicated in line 25 of BC3204C3M.
- . C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR (instead of CONSTRAINT_ERROR).
- . C41404A: The values of 'LAST and 'LENGTH are incorrect in IF statements from line 74 to the end of the test.
- . C48008A: This test requires that the evaluation of default initial values not occur when an exception is raised by an allocator. However, the Language Maintenance Committee (LMC) has ruled that such a requirement is incorrect (AI-00397/01).
- . C32114A: An unterminated string literal occurs at line 62.
- . B33203C: The reserved word "IS" is misspelled at line 45.
- . C34018A: The call of function G at line 114 is ambiguous in the presence of implicit conversions and inconsistent without.
- . B37401A: The object declarations at lines 126-135 follow subprogram bodies declared in the same declarative part.
- . B45116A: ARRPRIBL1 and ARRPRIBL2 are initialized with a value of the wrong type (PRIBOOL_TYPE instead of ARRPRIBOOL_TYPE) at line 41.
- . B49006A: Object declaratives at lines 41 and 50 are terminated incorrectly with colons; "END CASE;" is missing from line 42.

- . B74101B: The "BEGIN" at line 9 is mistaken; it causes the declarative part to be treated as a sequence of statements.
- . C87B50A: The call of "/"= at line 31 requires a "USE" clause for package A.
- . C92005A: At line 40, "/"= for type PACK.BIG_INT is not visible without a "USE" clause for package PACK.
- . C940ACA: This test assumes that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program; however, such an execution order is not required by the Ada Standard, so the test is erroneous.
- . CA3005A..D (4 tests): No valid elaboration order exists for these tests.

END OF LIST